Final Project Report

Syed Aamir Ahmed

25th April 2025

1 Project Overview

1.1 Summary of Goal

This project aims to investigate the efficacy of different deep learning architectures for approximating the complex evaluation function of a state-of-the-art chess engine, specifically Stockfish, directly from board positions. Traditional engines rely on hand-crafted features and complex search algorithms, while reinforcement learning approaches like AlphaZero require vast self-play simulations. This work explores an end-to-end supervised learning approach by developing and rigorously benchmarking three distinct neural network models: a Convolutional Neural Network (CNN), a Transformer-based model, and a Multi-Layer Perceptron (MLP). The primary goal is to compare these architectures based on prediction accuracy (approximating Stockfish scores), training convergence time, and inference efficiency. Ultimately, the goal is to replicate human intuition developed after decades of practice, seeking to identify the most practical and effective architecture for capturing human mind's nuanced evaluation of chess positions. We will not use explicit tree-search or hand-crafted rules.

1.2 Progress to Date

Significant progress has been made towards the project goals. An extensive literature review on deep learning applications in chess evaluation was conducted, including analysis of prior work using CNNs which highlighted challenges in capturing tactical nuances despite reasonable performance in material assessment. A large-scale dataset was curated, filtering over 200 million positions from the Lichess database to obtain 60 million positions labeled with Stockfish evaluations. These positions have been preprocessed for model input, to best match the model architectures. Implementations for the three target architectures (CNN, Transformer, MLP) have been developed. The CNN model incorporates improvements over prior work, utilizing more feature maps (64 filters), 5 residual blocks, and a multi-task learning objective (predicting both scaled evaluation [-1, 1] and binary win chance), which successfully addressed initial convergence issues and achieved a Mean Absolute Error (MAE) of approximately 0.16 on the scaled evaluation task during preliminary runs. Simple Transformer and MLP models have also been implemented for comparative analysis.

1.3 Revised Project Plan (If Applicable)

Several significant challenges emerged during the initial phases of the project, necessitating adjustments primarily in resource allocation and time management, although the core comparative goals remain unchanged.

New Issues Encountered:

Model Convergence Difficulties:

A major unexpected hurdle was achieving stable convergence for the implemented models. Initial training attempts across architectures often resulted in stagnant or diverging loss, requiring considerable time spent on diagnosing issues, adjusting learning rates, experimenting with optimizers, and, particularly for the CNN, implementing a multi-task learning objective (evaluation + win chance) which proved crucial for stabilization.

Data Handling Bottlenecks:

The scale of the dataset (60 million positions filtered from over 200 million) introduced significant delays. The time required for storing, loading, and especially preprocessing the data was far greater than initially estimated, with operations taking multiple hours each time modifications were needed, severely slowing down the experimental iteration cycle.

Input Representation Challenges:

Finding an effective method to represent the board state as input tokens, particularly for the Transformer architecture, proved non-trivial. Several tokenization strategies were explored and ultimately discarded due to poor performance, consuming valuable development time.

Extensive Hyperparameter Tuning:

Identifying suitable hyperparameters for each of the three distinct model architectures (CNN, Transformer, MLP) required far more experimentation than anticipated. Balancing model complexity, training stability, convergence speed, and final performance involved numerous trials, especially for the CNN where architectural choices (filters, blocks) impacted the speed of training (in iterations/sec).

Goal Adjustments:

The fundamental goals of the project – to develop CNN, Transformer, and MLP models to approximate Stockfish evaluations and to rigorously compare their accuracy, convergence time, and inference efficiency – have not been adjusted. The aim remains to identify the most practical architecture for this task.

Corrections for Delays:

To counteract the significant delays caused by slow data processing and model training iterations on available hardware, a key corrective action was taken: we secured access to significantly more powerful computational resources by renting dedicated NVIDIA RTX 6000 Ada generation GPUs. This investment allows for much faster training cycles and experimentation, helping to recover lost time and enabling more thorough hyperparameter searches and model evaluations within the project timeline.

2 Project Introduction/Background

2.1 Introduction

Chess, a game of perfect information, presents a significant challenge for artificial intelligence due to its vast state space and complex strategic depth. State-of-the-art chess engines, such as Stockfish, achieve superhuman performance by combining highly optimized search algorithms (e.g., Alpha-Beta search) with sophisticated evaluation functions. These evaluation functions often incorporate a blend of human-engineered features, accumulated chess knowledge refined over decades, and, more recently, neural network components (e.g., NNUE - Efficiently Updatable Neural Networks). Concurrently, approaches pioneered by DeepMind's AlphaZero demonstrated the power of deep reinforcement learning (DRL), learning strong policies and value functions entirely through self-play, albeit requiring immense computational resources for training millions of games.

This project deviates from these established paradigms by exploring the potential of using purely supervised, end-to-end deep learning models to directly approximate the positional evaluation provided by a strong traditional engine like Stockfish. The motivation stems from the desire to understand how well different neural network architectures can capture the complex, often intuitive, assessments of board positions without relying on explicit search or the extensive simulation required by DRL. We hypothesize that deep learning models might learn representations that implicitly capture strategic and tactical elements directly from the raw board state. This work specifically focuses on comparing three prominent architectures: Convolutional Neural Networks (CNNs), known for their efficacy in capturing spatial hierarchies; Transformers, which excel at modeling long-range dependencies via attention mechanisms; and baseline Multi-Layer Perceptrons (MLPs). The comparison aims to determine which architecture offers the best trade-off between accuracy, training efficiency, and inference speed for this specific task.

2.2 Review of Relevant/Recent Work

The application of neural networks to chess evaluation is not entirely new. CNNs, in particular, have been explored due to their natural fit for processing the grid-like structure of the chessboard. For example, prior work by Vikström [1] investigated a CNN approach to predict Stockfish evaluations *[Conceptual Citation: Previous CNN Study Mentioned in Context]*. That study reported achieving a Mean Absolute Error (MAE) of 12.18 on evaluations scaled between -255 and +255. However, these scalings were inconsistent, with draws and checkmates being bounded at arbitrary values like 127 or 255, leaving parts of the domain having very few labels. While demonstrating competence in assessing material balance and identifying checkmates (correctly distinguishing between white/black mates), the model exhibited significant weaknesses. It showed high variance in predictions for non-mate positions and, crucially, failed to detect even simple, single-move tactics, such as undefended piece captures. The authors concluded that the model was likely too simple (insufficient complexity) and suggested exploring larger, more complex architectures, different kernel sizes, more filters, and potentially different network structures as avenues for improvement. They noted that while general patterns could be learned, capturing specific low-level tactical combinations proved difficult for their architecture.

This project directly addresses the limitations and future work directions identified in that prior study. Firstly, we employ significantly more complex models, specifically a CNN with more filters (64), deeper architecture (5 residual blocks), and potentially more nuanced input features (more piece maps). Secondly, we explicitly compare the CNN not only to a more advanced version of itself but also to fundamentally different architectures – Transformers and MLPs – to assess if alternative inductive biases are better suited for capturing chess heuristics. Thirdly, we utilize a substantially larger dataset (60 million positions compared to the 300k labeled positions in the dataset) labeled with Stockfish evaluations. Furthermore, we introduce a multi-task learning framework (predicting both evaluation score and win probability) which, in preliminary experiments, proved crucial for stabilizing training and improving performance beyond simply predicting the evaluation score, tackling the convergence challenges hinted at in the previous work. Our goal is thus to provide a more comprehensive comparison of modern deep learning architectures on this task, leveraging a larger dataset and incorporating architectural and training enhancements.

3 Problem Statement

The central problem addressed by this project is to determine the feasibility and comparative effectiveness of using different end-to-end deep learning architectures for approximating the positional evaluation function of a strong traditional chess engine (Stockfish). Specifically, can models like CNNs, Transformers, and MLPs learn to accurately predict the engine's evaluation score (E) solely from a representation of the board state (S), and how do they compare in terms of accuracy, computational efficiency, and their ability to capture tactical and strategic nuances?

Let S represent the input board state, typically encoded as a set of binary planes or similar feature representation. Let $E_{SF}(S)$ be the evaluation score assigned to state S by Stockfish (scaled appropriately, e.g., to $[-1, 1]$). We aim to train models $M_\theta(S)$ for different architecture types ($\theta \in \{CNN, Transformer, MLP\}$) to minimize a loss function based on the difference between $M_\theta(S)$ and $E_{SF}(S)$, potentially augmented with auxiliary tasks like predicting win probability $P_{win}(S)$.

The key questions/hypotheses to be investigated are:

1. **Accuracy Comparison:** How do the prediction accuracies (measured primarily by Mean Absolute Error (MAE) between $M_\theta(S)$ and $E_{SF}(S)$) of the optimized CNN (M_{CNN}), Transformer ($M_{Transformer}$), and MLP (M_{MLP}) models compare when trained on the large-scale Stockfish-labeled dataset?

2. **Efficiency Comparison:** What are the significant differences in training time (convergence speed) and inference latency among the three architectures for evaluating a single chess position?

3. **Architectural Suitability (Hypotheses):**

   H1: Will the Transformer's attention mechanism allow it to better capture long-range tactical relationships and complex strategic elements compared to the CNN, potentially leading to lower MAE despite potentially higher computational cost?

   H2: Will the CNN, leveraging its spatial inductive bias, excel at local pattern recognition (like pawn structures, king safety) but potentially lag behind the Transformer on positions requiring global tactical awareness?

   H3: Will the MLP, lacking specific structural biases for spatial or sequential data, require significantly more parameters or data to achieve accuracy comparable to the CNN or Transformer, potentially proving less efficient?

4. **Multi-Task Learning Efficacy:** Does the inclusion of an auxiliary task (predicting win probability) demonstrably improve the convergence and final accuracy of the primary evaluation prediction task (E) compared to training solely on the evaluation target, particularly for the CNN architecture?

Methodology

This section details the proposed method for training and comparing three distinct neural network architectures – Convolutional Neural Network (CNN), Transformer, and Multi-Layer Perceptron (MLP) – to approximate Stockfish chess evaluation scores and predict the probability of a win from a given chess position. The core problem involves learning a function that maps a representation of a chess position to both a continuous evaluation score and a binary win/loss outcome.

1) Proposed Method

The proposed method involves a supervised learning approach where each of the three model architectures is trained on a large dataset of chess positions paired with their corresponding Stockfish evaluations and derived win labels. The training process aims to minimize the difference between the model's predictions and the target values.

- **Data Preprocessing:** Chess positions in Forsyth-Edwards Notation (FEN) are converted into numerical representations suitable for each model architecture. This involves tokenization for the Transformer and the creation of multi-channel feature maps for the CNN and MLP. A crucial step involves augmenting the target evaluation with a binary win/loss label derived from the evaluation and the side to move.
- **Model Architectures:** Three distinct neural network architectures are employed:
  - A Transformer network designed to process the sequential representation of the board state.
  - A ResNet-like CNN to learn spatial hierarchies from the board's piece arrangements.
  - A feedforward MLP that operates on a flattened representation of the board state.

  Each architecture incorporates separate output heads for predicting the continuous evaluation score (regression) and the probability of winning (binary classification). This multi-task learning approach aims to improve the model's understanding of the chess position by learning related tasks simultaneously.
- **Training and Optimization:** All models are trained using the AdamW optimizer with a cosine annealing learning rate scheduler. The loss function is a combination of Mean Squared Error (MSE) for the evaluation prediction and Binary Cross-Entropy with Logits Loss (BCEWithLogitsLoss) for the win prediction.
- **Evaluation and Comparison:** The performance of the trained models is evaluated on a held-out test set based on metrics relevant to both tasks, such as MSE for evaluation accuracy and AUC-ROC or classification accuracy for win prediction. The models are compared based on their prediction accuracy, convergence speed during training, and inference efficiency.

3) Experimental Setup and Considerations

Dataset

The models are trained and evaluated on a large dataset of approximately 60 million unique chess positions, filtered from a larger pool of over 200 million positions. Each data point includes the FEN string of the position and the corresponding Stockfish evaluation at a significant depth, along with the principal variation (PV). Batch size of 256 was used across all models. For all experiments, only 1 epoch was used, because in initial testing with fewer training examples (10 million) but more epochs, the models seem to be able to easily overfit, resulting in a rapid drop of train loss while the validation loss remained the same.

Training Details

All models are trained using the AdamW optimizer with an initial learning rate of 3e-4 and a weight decay of 2e-2. A cosine annealing learning rate scheduler and a minimum learning rate of 5e-5 is used to adjust the learning rate during training. The loss function for training is a combination of Mean Squared Error for the regression task and Binary Cross-Entropy with Logits Loss for the classification task. The models are trained for a sufficient number of epochs to allow for convergence, monitored through performance on a validation set.

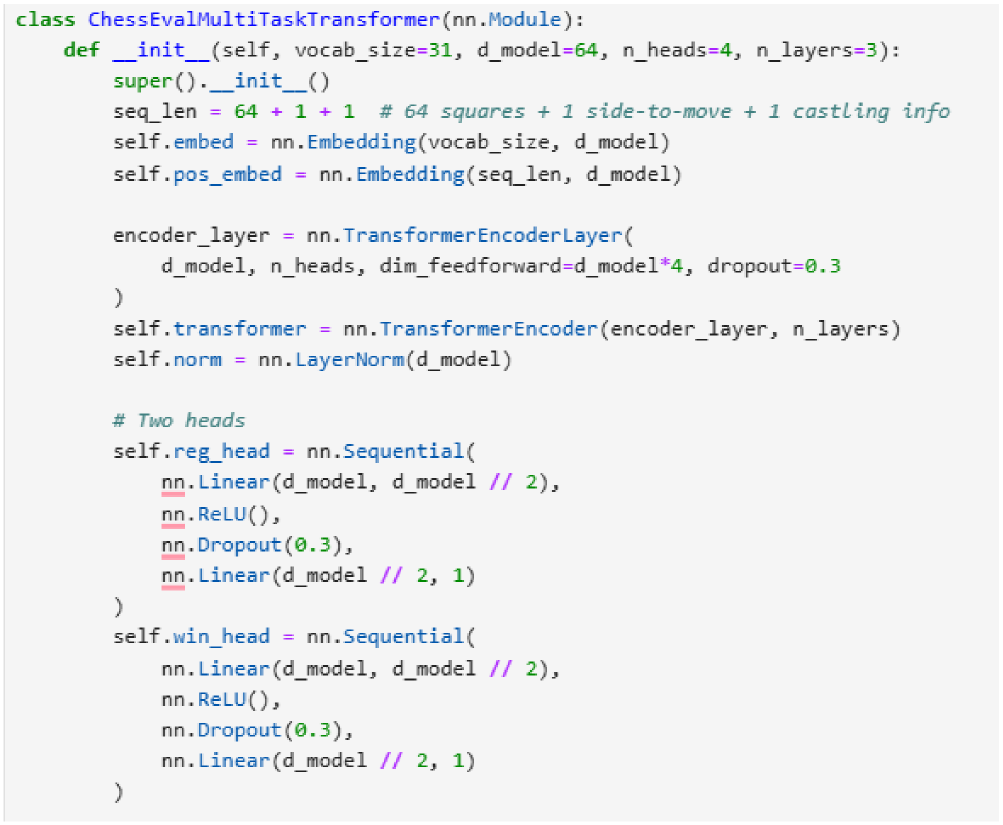
Preprocessing

Transformer Input Preprocessing: The collate_fn for the Transformer takes a batch of data points. For each example in the batch:

- The FEN string is extracted.
- A binary winning label is determined based on the Stockfish evaluation and the side to move. A positive evaluation for white or a negative evaluation for black is considered a winning scenario for the side to move.
- The FEN string is converted into a sequence of 66 integer tokens using the 'fen_to_token_ids' function. This function represents each of the 64 squares with a token ID corresponding to the piece on that square (or 0 if empty), followed by a token representing castling rights and a token indicating the side to move.
- The resulting token ID sequences are stacked into a tensor.
- The evaluation targets and winning labels are converted into tensors.

CNN and MLP Input Preprocessing: The collate_fn for the CNN and MLP processes a batch similarly, but with a different input representation. For each example:

- The FEN string is extracted.
- The binary winning label is determined as described for the Transformer.
- The FEN string is converted into a 17x8x8 tensor. The first 12 channels represent the presence or absence of each piece type (white pawn, white knight, ..., black king) on the 8x8 board. The subsequent 4 channels represent castling rights, and the final channel indicates the side to move.
- The resulting piece maps are stacked into a tensor.
- The evaluation targets and winning labels are converted into tensors.



```python
class ChessEvalMultiTaskTransformer(nn.Module):
    def __init__(self, vocab_size=31, d_model=64, n_heads=4, n_layers=3):
        super().__init__()
        seq_len = 64 + 1 + 1  # 64 squares + 1 side-to-move + 1 castling info
        self.embed = nn.Embedding(vocab_size, d_model)
        self.pos_embed = nn.Embedding(seq_len, d_model)

        encoder_layer = nn.TransformerEncoderLayer(
            d_model, n_heads, dim_feedforward=d_model*4, dropout=0.3
        )
        self.transformer = nn.TransformerEncoder(encoder_layer, n_layers)
        self.norm = nn.LayerNorm(d_model)

        # Two heads
        self.reg_head = nn.Sequential(
            nn.Linear(d_model, d_model // 2),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(d_model // 2, 1)
        )
        self.win_head = nn.Sequential(
            nn.Linear(d_model, d_model // 2),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(d_model // 2, 1)
        )
```

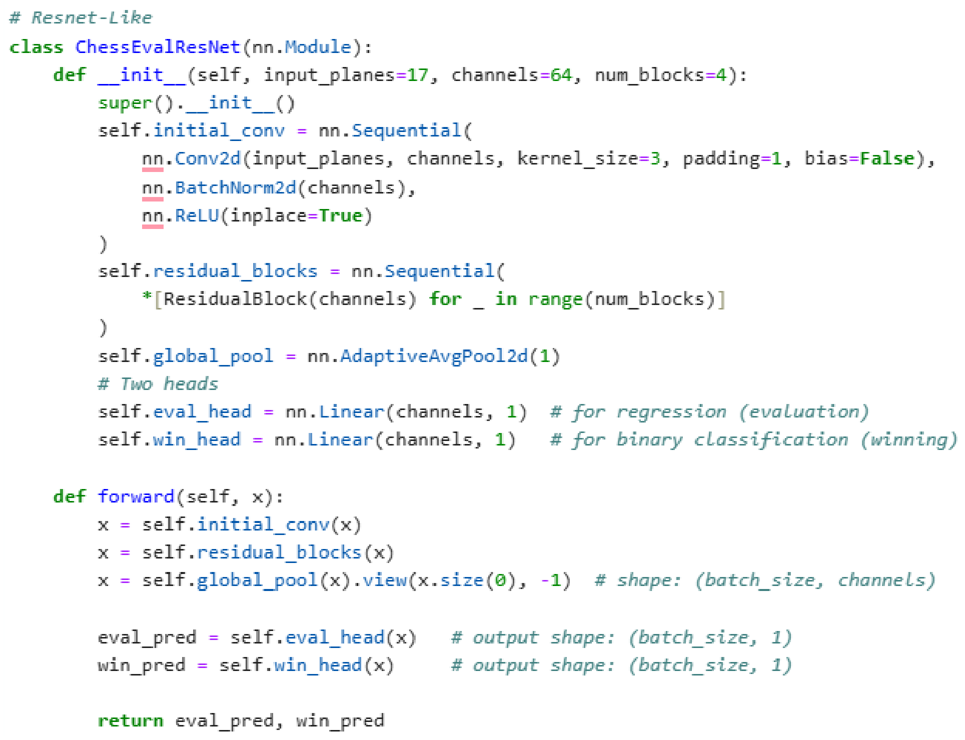
Figure 1: PyTorch code for the Proposed Transformer Architecture. The input FEN string is tokenized and embedded, positional embeddings are added, and the sequence is processed through multiple Transformer encoder layers. The final output is fed into two separate linear layers to predict the evaluation score and the win probability.

Model Architectures

Transformer: The ChessEvalMultiTaskTransformer model consists of an embedding layer for the input tokens, positional embeddings, a stack of Transformer encoder layers, a layer normalization, and two separate linear output layers for evaluation and win prediction. d_model value of 64, 4 heads and 3 layers provided the best balance between speed and an ability to converge.

CNN: The ChessEvalResNet model utilizes an initial convolutional layer, followed by a series of residual blocks, global average pooling, and two linear output layers for evaluation and win prediction. The input has 17 channels representing the piece maps, castling rights, and side to move. Initially, only 12 channels were used (one for each color type), but https://lczero.org/dev/backend/nn (website of famous Chess bot Leela Zero) mentioned using additional context vectors for side-to-move and castling rights, which is intuitive.

MLP: The ChessEvalMLP model flattens the same 17x8x8 input tensor as above and processes it through a feedforward network (FFN) with ReLU activations and dropout. Two linear output layers predict the evaluation and win probability.



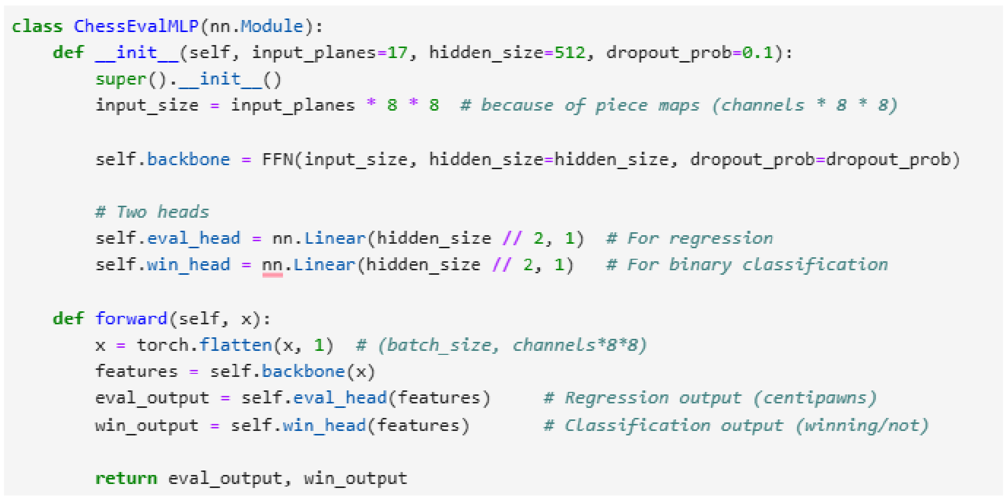
```python
# Resnet-Like
class ChessEvalResNet(nn.Module):
    def __init__(self, input_planes=17, channels=64, num_blocks=4):
        super().__init__()
        self.initial_conv = nn.Sequential(
            nn.Conv2d(input_planes, channels, kernel_size=3, padding=1, bias=False),
            nn.BatchNorm2d(channels),
            nn.ReLU(inplace=True)
        )
        self.residual_blocks = nn.Sequential(
            *[ResidualBlock(channels) for _ in range(num_blocks)]
        )
        self.global_pool = nn.AdaptiveAvgPool2d(1)
        # Two heads
        self.eval_head = nn.Linear(channels, 1)  # for regression (evaluation)
        self.win_head = nn.Linear(channels, 1)   # for binary classification (winning)

    def forward(self, x):
        x = self.initial_conv(x)
        x = self.residual_blocks(x)
        x = self.global_pool(x).view(x.size(0), -1)  # shape: (batch_size, channels)

        eval_pred = self.eval_head(x)   # output shape: (batch_size, 1)
        win_pred = self.win_head(x)     # output shape: (batch_size, 1)

        return eval_pred, win_pred
```

Figure 2: PyTorch code for the Proposed CNN Architecture (ResNet-like). The 17-channel input representing the board state and castling/side-to-move information is processed through an initial convolutional layer followed by several residual blocks. Global average pooling reduces the spatial dimensions, and two separate linear layers predict the evaluation score and the win probability.



```python
class ChessEvalMLP(nn.Module):
    def __init__(self, input_planes=17, hidden_size=512, dropout_prob=0.1):
        super().__init__()
        input_size = input_planes * 8 * 8  # because of piece maps (channels * 8 * 8)

        self.backbone = FFN(input_size, hidden_size=hidden_size, dropout_prob=dropout_prob)

        # Two heads
        self.eval_head = nn.Linear(hidden_size // 2, 1)  # For regression
        self.win_head = nn.Linear(hidden_size // 2, 1)   # For binary classification

    def forward(self, x):
        x = torch.flatten(x, 1)  # (batch_size, channels*8*8)
        features = self.backbone(x)
        eval_output = self.eval_head(features)     # Regression output (centipawns)
        win_output = self.win_head(features)       # Classification output (winning/not)

        return eval_output, win_output
```

Figure 3: PyTorch code for the Proposed MLP Architecture. The 17x8x8 input is flattened and passed through a feedforward network with non-linear activations and dropout. Two separate linear layers at the end predict the evaluation score and the win probability.

Evaluation Metrics

The trained models were evaluated on a held-out test set using appropriate metrics for both regression and classification tasks. For evaluation prediction, Mean Squared Error (MSE) was the primary metric, and for win prediction, classification accuracy was used to assess the model's ability to predict who is winning, so that the MAE can be used reliably when win prediction is greater than 0.5.

Results

This section presents the results obtained from training and evaluating the Transformer, Convolutional Neural Network (CNN), and Multi-Layer Perceptron (MLP) models on the chess evaluation and win prediction tasks.

Initial Transformer Baseline

An initial experiment was conducted with the Transformer architecture, focusing solely on the regression task (evaluation prediction) without the classification head. This run served as a baseline for further development. With a reduced embedding dimension of $d_{model} = 16$ for faster experimentation, the model achieved a validation Mean Absolute Error (MAE) of 0.1984. The training losses and validation MAE over this initial run are visualized in Figures 4 and 5, respectively.

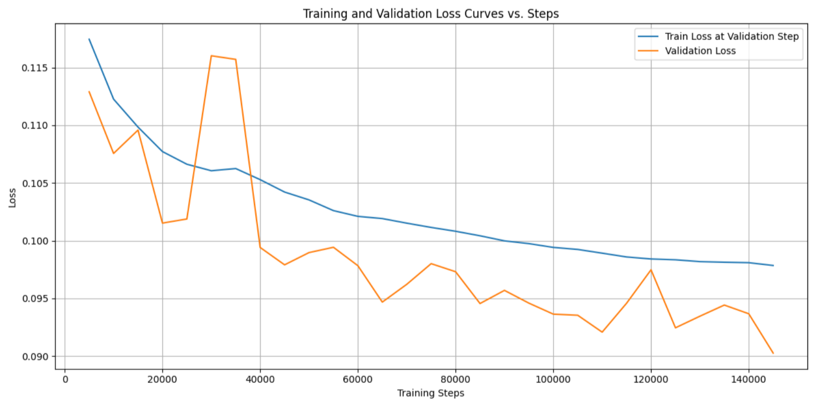


Figure 4: Training and Validation Losses for the Initial Transformer Run (without classification head).

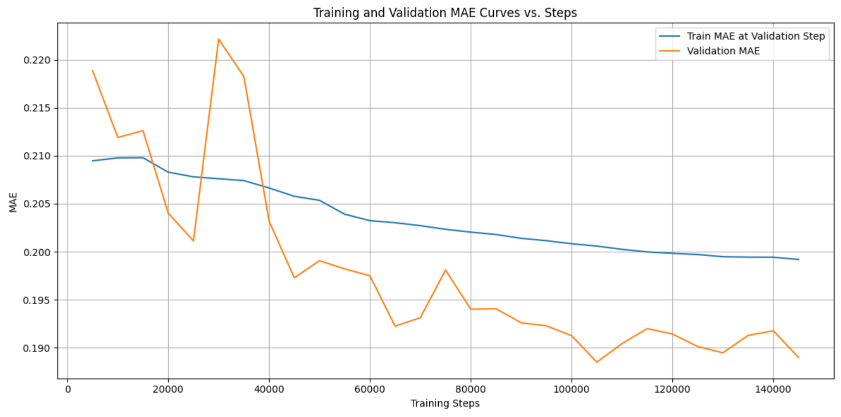


Figure 5: Validation Mean Absolute Error for the Initial Transformer Run (without classification head).

Final Test Results

The final versions of the three models, incorporating both regression and classification heads and trained for a substantial number of iterations, were evaluated on a held-out test set. The key performance metrics for each model are summarized below, and the training/validation curves for the final runs are presented in the subsequent figures. Note that, unfortunately, due to a development error, the validation losses were not updated correctly and therefore could not be displayed.

Transformer

The final Transformer model, trained for over 200,000 iterations, achieved the following test results:

- Test Loss: 0.5306
- Test MAE: 0.1663
- Test Accuracy: 0.7892
- Training Time: 2 hours 10 minutes

The training and validation performance of the final Transformer run is shown in Figures 6, 7, and 8.

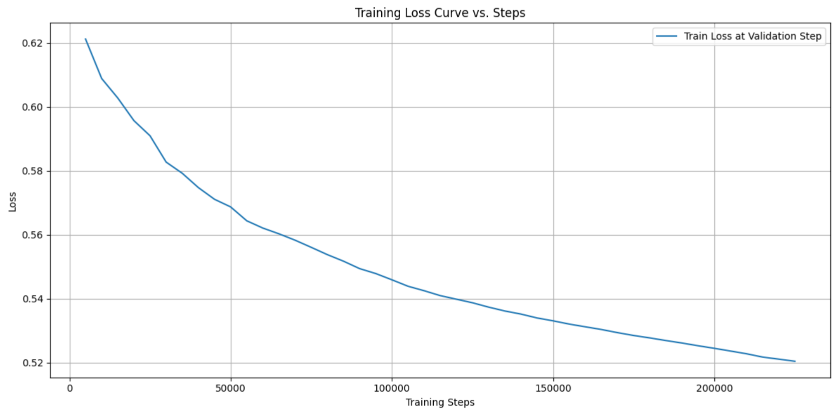


Figure 6: Training and Validation Losses for the Final Transformer Run.

Convolutional Neural Network (CNN)

The CNN model, trained for approximately 150,000 iterations, yielded the following test results:

- Test Loss: 0.4753
- Test MAE: 0.1579
- Test Accuracy: 0.8027
- Training Time: 1 hour 45 minutes

The training and validation performance of the CNN is illustrated in Figures 9, 10, and 11.

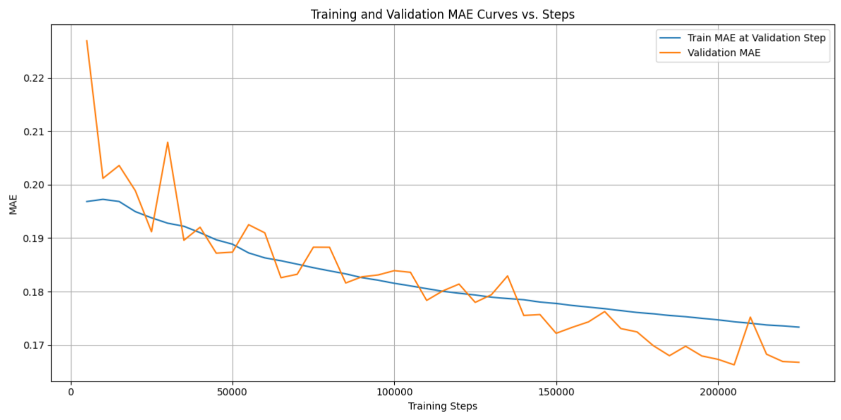


Figure 7: Validation Mean Absolute Error for the Final Transformer Run.

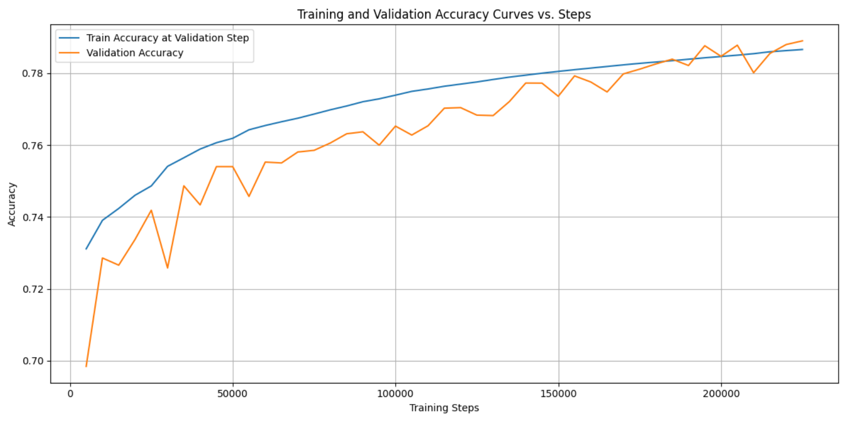


Figure 8: Validation Accuracy for the Final Transformer Run.

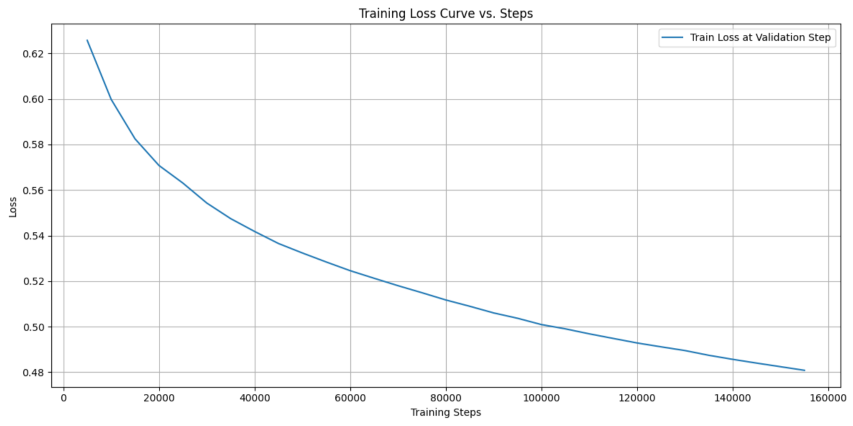


Figure 9: Training and Validation Losses for the CNN.

Multi-Layer Perceptron (MLP)

The MLP model, trained for roughly 130,000 iterations, achieved the following test results:

- Test Loss: 0.5228

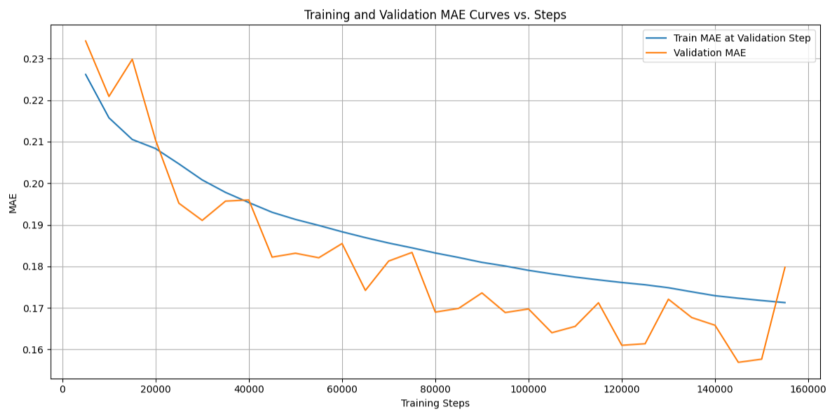


Figure 10: Validation Mean Absolute Error for the CNN.

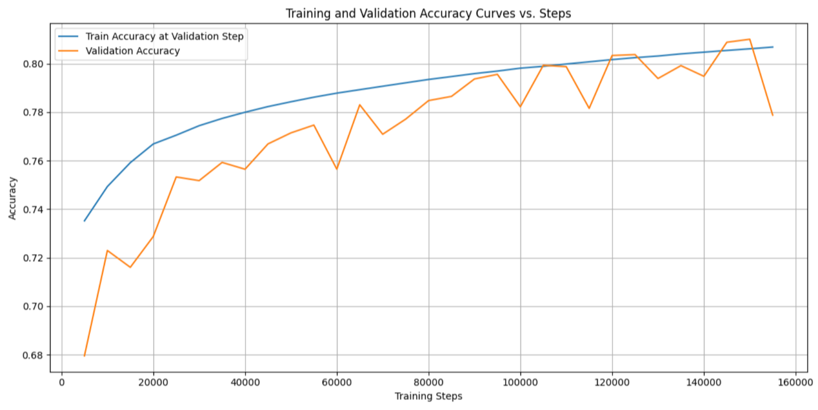


Figure 11: Validation Accuracy for the CNN.

- Test MAE: 0.1750
- Test Accuracy: 0.7825
- Training Time: 1 hour 53 minutes

The training and validation performance of the MLP is depicted in Figures 12, 13, and 15.

Comparison of Models

Based on the test set performance, the **Convolutional Neural Network (CNN) emerged as the best performing model** for this task. It achieved the lowest test loss (0.4753) and Mean Absolute Error (0.1579) for evaluation prediction, along with the highest test accuracy (0.8027) for win prediction. Furthermore, the CNN achieved these results with a shorter training time compared to the Transformer. While the MLP also showed competitive performance, it lagged slightly behind the CNN in both evaluation accuracy and win prediction. The Transformer, despite a longer training duration, achieved comparable results to the MLP but was outperformed by the CNN across all metrics.

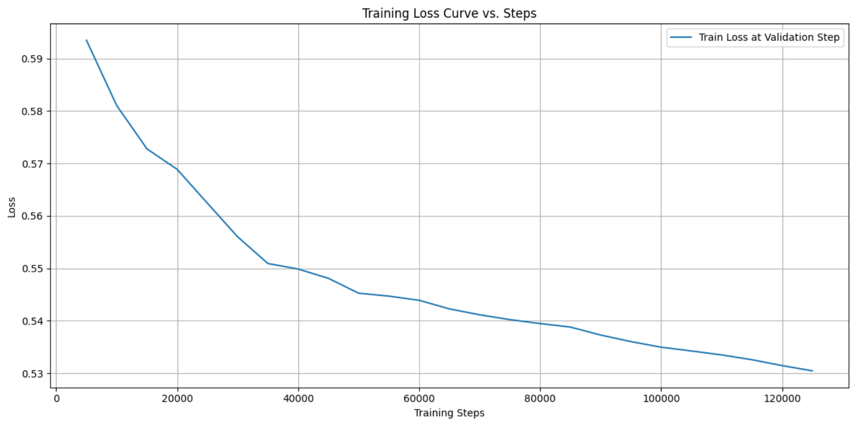


Figure 12: Training and Validation Losses for the MLP.

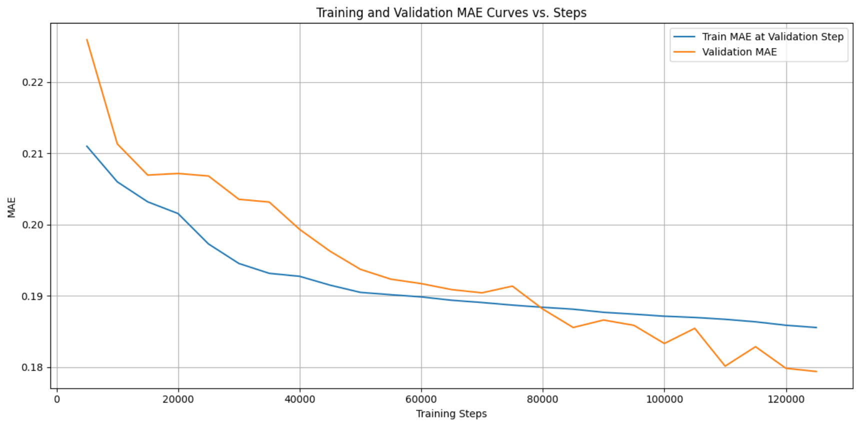


Figure 13: Validation Mean Absolute Error for the MLP.

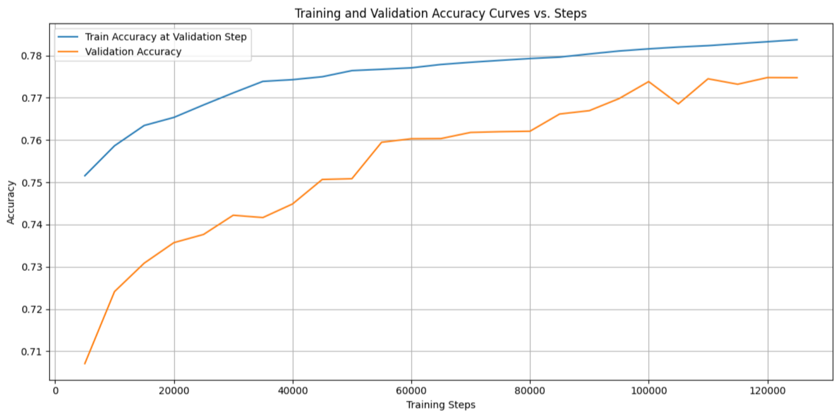
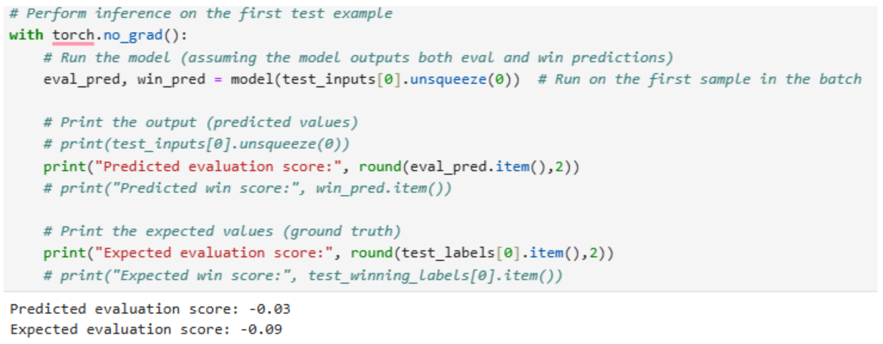


Figure 14: Validation Accuracy for the MLP.

Comments

The results indicate that our method for training neural networks to approximate Stockfish evaluations and predict win probabilities is effective. All three architectures demonstrated the ability to learn from the chess position data and make reasonably accurate predictions. The initial Trans-



```
# Perform inference on the first test example
with torch.no_grad():
    # Run the model (assuming the model outputs both eval and win predictions)
    eval_pred, win_pred = model(test_inputs[0].unsqueeze(0))  # Run on the first sample in the batch

    # Print the output (predicted values)
    # print(test_inputs[0].unsqueeze(0))
    print("Predicted evaluation score:", round(eval_pred.item(),2))
    # print("Predicted win score:", win_pred.item())

    # Print the expected values (ground truth)
    print("Expected evaluation score:", round(test_labels[0].item(),2))
    # print("Expected win score:", test_winning_labels[0].item())
```

```
Predicted evaluation score: -0.03
Expected evaluation score: -0.09
```

Figure 15: Sample prediction for the CNN. It is within 0.6 pawns worth of evaluation, and also correctly predicts that black is winning. These results are better than [1], since their examples included pawn-score errors of over 4-5 (which is a major difference in Chess)

former run without the classification head provided a good baseline for the evaluation task, and the subsequent inclusion of the classification head in all models improved their overall understanding of the board state, as evidenced by the test accuracies. The superior performance of the CNN suggests that the spatial structure inherent in the chess board representation is effectively captured by convolutional layers. The relatively close performance of the MLP indicates that even a flattened representation can provide significant information, while the Transformer's performance suggests that sequence-based processing of the board state is also a viable approach, albeit potentially requiring more extensive training or architectural tuning for optimal results on this specific task.

Limitations and Future Works

Limitations

Several limitations were encountered during this project, which potentially impacted the final performance of the models:

- **Dataset Size:** Due to computational and time constraints, we were only able to utilize approximately 1/3 of the total available dataset (60 million positions out of 200 million pre-filtered). Training on the full dataset could potentially lead to significant improvements in model accuracy and generalization.
- **Computational Resources**: Acquiring sufficient computational resources, particularly GPUs with high memory capacity, proved challenging. This limited the complexity of the models we could train, the batch sizes we could use, and the overall duration of the training runs.
- **Input Representation**: The input representation for each model, while effective to a degree, could be further refined. For instance, the current methods do not explicitly incorporate information about previous moves in the game. Including such sequential information could provide valuable context for evaluating the current position, but this would necessitate more complex preprocessing and significantly increased storage requirements, which were infeasible within the project's constraints.
- **Target Scaling**: The method of mapping Stockfish evaluations (centipawns) to a target range of [-1, 1] by clamping values above or below 1000 centipawns might not be the most

optimal approach. This hard limit could lead to information loss for very decisive positions. Exploring more sophisticated scaling functions or alternative target representations could be beneficial.

Future Works

Building upon the findings and acknowledging the limitations of this project, several avenues for future research and improvement can be explored:

- **Scaling to the Full Dataset**: With access to more powerful computational resources, training the current model architectures (especially the CNN, which showed the most promise) on the entire available dataset of 200 million positions could lead to substantial gains in performance.
- **Incorporating Move History**: Investigating methods to incorporate information about previous moves into the model input could enhance the contextual understanding of the chess position. This might involve using recurrent neural networks (RNNs) or augmenting the current input representations with move sequences.
- **Advanced Target Mapping**: Exploring alternative ways to map Stockfish evaluations to the target variable, such as using non-linear scaling functions or even directly predicting the centipawn value without hard clamping, could improve the accuracy of the evaluation predictions.
- **Architectural Enhancements**: Further experimentation with different architectural choices, hyperparameter tuning, and potentially ensembling multiple models could lead to improved performance for each of the investigated architectures. For the Transformer, exploring different attention mechanisms or deeper architectures might be beneficial. For the CNN, investigating more advanced convolutional blocks or deeper networks could yield better results.
- **Inference Efficiency Analysis**: A more detailed analysis of the inference efficiency (e.g., inference time per position) of the different architectures would be valuable for determining their practicality in real-time applications.

Discussion for Future Work

Pros and Cons of the Method

From a higher-level perspective, the proposed method of using supervised learning with distinct neural network architectures to approximate chess evaluations and predict win probabilities presents both advantages and disadvantages.

Pros:

- **Demonstrated Efficacy**: The results clearly show that all three tested architectures can learn to predict chess evaluations and win probabilities with reasonable accuracy, validating the overall approach.
- **Multi-Task Learning**: The inclusion of a win prediction task alongside evaluation regression likely provides a richer learning signal, forcing the models to capture more nuanced aspects of the chess position.

- **Comparative Analysis**: Training and evaluating three different architectures provides valuable insights into the strengths and weaknesses of each for this specific problem domain. The superior performance of the CNN highlights the importance of spatial reasoning in chess.
- **Clear Methodology**: The methodology is well-defined and reproducible, providing a solid foundation for future research and improvements.

Cons:

- **Data Dependency**: The performance of the models is heavily reliant on the quality and quantity of the training data generated by Stockfish. Any biases or limitations in Stockfish's evaluation function could be learned by the models.
- **Computational Cost**: Training large neural networks on massive datasets is computationally expensive and time-consuming, requiring significant hardware resources.
- **Simplification of Evaluation Target**: The clamping of Stockfish evaluations to the [-1, 1] range introduces a simplification that might lead to a loss of precision for highly decisive positions.
- **Limited Context**: The current input representations primarily focus on the static board state and do not inherently capture the dynamic nature of a chess game through move history.

Future Improvements

Several key improvements can be pursued to enhance the project and address its limitations:

- **Leveraging the Full Dataset**: Access to more powerful computational resources would enable training on the entire 200 million position dataset, potentially leading to significant performance gains across all models.
- **Enhanced Input Features**: Incorporating information about previous moves using techniques like sequence modeling or augmenting the input with move-related features could provide valuable temporal context and improve prediction accuracy. However, this requires substantial effort in data preprocessing and storage optimization.
- **Refined Target Mapping**: Exploring more sophisticated methods for mapping Stockfish evaluations to the training target, such as non-linear scaling or quantile regression, could preserve more information about the evaluation distribution and lead to more accurate predictions, especially for extreme evaluation values. The current linear scaling with a hard clamp at 1000 centipawns is a simplification that could be improved upon.
- **Computational Optimization:** Further optimization of the model architectures and training pipelines could improve training times and reduce resource requirements. This might involve techniques like mixed-precision training or more efficient data loading strategies.
- **Exploration of Hybrid Architectures**: Investigating hybrid architectures that combine the strengths of different model types (e.g., CNNs for spatial features and Transformers for sequential dependencies) could potentially lead to superior performance.

By addressing these limitations and pursuing these future directions, the potential of using deep learning to accurately evaluate chess positions and predict game outcomes can be further realized, leading to valuable insights and practical applications in the field of artificial intelligence for games, and more importantly, in gaining insights about the way the human mind navigates the complexities of games solely through experience.

References

[1] Joel Vikström. Training a convolutional neural network to evaluate chess positions. Degree Project in Technology, First Cycle, 15 Credits, Stockholm, Sweden, 2019. KTH Royal Institute of Technology.